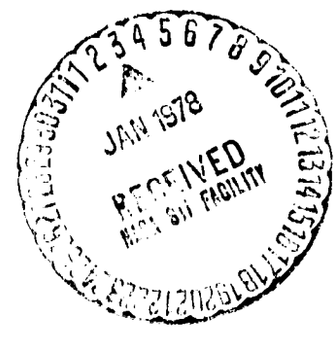
NASA TM-73,203

An ILLIAC Program for the Numerical Simulation of Homogeneous Incompressible Turbulence

Robert S. Rogallo

November 1977



National Aeronautics and Space Administration

Ames Research Center
Moffett Field, California 94035

AN ILLIAC PROGRAM FOR THE NUMERICAL SIMULATION OF HOMOGENEOUS INCOMPRESSIBLE TURBULENCE

Robert S. Rogallo

Ames Research Center

SUMMARY

An algorithm and ILLIAC computer program, developed for the simulation of homogeneous incompressible turbulence in the presence of an applied mean strain, are described. The turbulence field is represented spatially by a truncated triple Fourier series (spectral method) and followed in time using a fourth-order Runge-Kutta algorithm. Several transformations are applied to the numerical problem to enhance the basic algorithm. These include

1. Transformation of variables suggested by Taylor's sudden-distortion theory

2. Implicit viscous diffusion by use of an integrating factor

3. Implicit pressure calculation suggested by Taylor's sudden-distortion theory

4. Inexpensive control of aliasing by random and phased coordinate shifts

INTRODUCTION

The primary difficulty in the numerical simulation of homogeneous turbulence is that the nonlinearity of the equations of fluid motion excites a large range of scales (i.e., a large ratio of largest to smallest scale) of motion in both space and time. The computer resource required for a complete simulation is proportional to the product, over all space-time dimensions, of the range of computed scales of each dimension. These scale ranges increase with Reynolds number (R), and their product increases so rapidly, in three space dimensions, that only the weakest experimentally studied turbulence can be simulated completely on today's computers.

The overall range of scales continues to increase indefinitely with Reynolds number. However, at a sufficiently high Reynolds number the scales of motion can be grouped, in order of decreasing scale, into three distinct ranges: the energy-containing range, the "inertial" range, and the dissipation range (fig. 1). Further increases in Reynolds number increase only the inertial range. The range of energy-containing scales, which determine the

features of turbulence of engineering interest, is bounded as $R \to \infty$, and the motion in these scales becomes independent of the motion at smaller scales. At somewhat lower R, the inertial and dissipation ranges merge, but still do not affect the energy-containing range. At sufficiently low R, dissipation occurs in the energy-containing range itself. This physical description of the scale dependence upon Reynolds number is encouraging because it indicates that, in principle, only the energy-containing scales of motion need to be included in a high Reynolds number turbulence simulation. The difficulty is that, mathematically, all the scales are coupled through the nonlinear terms in the governing equations and, although we know that physically (i.e., statistically) the energy-containing range is uncoupled from the smaller scales, we do not know how to uncouple it mathematically.

The range of statistically interdependent scales increases with the anisotropy of the motion and, because most flows of engineering interest are anisotropic, it is important to determine the nature and magnitude of the additional computational difficulty posed by anisotropy.

THE NUMERICAL SIMULATION

The computational tool presented here is an unsteady incompressible Navier-Stokes code that runs on the ILLIAC IV computer. The program computes the evolution in time from an arbitrary homogeneous turbulence field in the presence of a simple class of spatially-linear mean flows. The simulation is a spectral decomposition similar to that of Orszag (ref. 1) but differing in detail. The primary purpose of this report is to present the simulation algorithm in detail sufficient to allow its use by others. The program can be used as presented to study weak (low Reynolds number) turbulence for which typical results are presented. The magnitude of the computation (fig. 2) requires a computer at least as fast as a CDC-7600.

THE EQUATIONS OF MOTION

The equations governing the flow of a viscous constant-density fluid are the familiar Navier-Stokes equations

$$\left.\begin{aligned} u_t + (uu)_x + (vu)_y + (wu)_z + p_x &= \nu(u_{xx} + u_{yy} + u_{zz}) \\ v_t + (uv)_x + (vv)_y + (wv)_z + p_y &= \nu(v_{xx} + v_{yy} + v_{zz}) \\ w_t + (uw)_x + (vw)_y + (ww)_z + p_z &= \nu(w_{xx} + w_{yy} + w_{zz}) \\ u_x + v_y + w_z &= 0 \end{aligned}\right\} \qquad (1)$$

where (u,v,w) is the velocity vector, p is the pressure-density ratio, ν is the kinematic viscosity, and subscripts denote differentiation.

We wish to simulate numerically the effect of a simple class of imposed strains on a homogeneous field of turbulence. The strain field is given by

$$(\bar{u},\bar{v},\bar{w}) = [xa(t),yb(t),zc(t)]$$

where $a + b + c = 0$ as required by continuity. It is convenient to introduce the following transformation of the dependent variables:

$$\left.\begin{aligned} u &= ax + A^{1/2}\hat{u} \\ v &= by + B^{1/2}\hat{v} \\ w &= cz + C^{1/2}\hat{w} \\ p &= -\frac{1}{2}\left[\left(\frac{da}{dt} + a^2\right)x^2 + \left(\frac{db}{dt} + b^2\right)y^2 + \left(\frac{dc}{dt} + c^2\right)z^2\right] + \hat{p} \end{aligned}\right\} \qquad (2)$$

where $a(t)$, $b(t)$, and $c(t)$ are the arbitrary time-dependent strain rates imposed, and the resulting inverse square strains are

$$\left.\begin{aligned} A(t) &= e^{-2\int_o^t a\,dt} \\ B(t) &= e^{-2\int_o^t b\,dt} \\ C(t) &= e^{-2\int_o^t c\,dt} \end{aligned}\right\} \qquad (3)$$

It follows from the continuity condition that material volumes are invariant, (i.e., $ABC = 1$). Explicit spatial dependence of the resulting system of equations is eliminated by the following transformation of independent variables:

$$\left.\begin{aligned} \hat{x} &= A^{1/2}x \\ \hat{y} &= B^{1/2}y \\ \hat{z} &= C^{1/2}z \end{aligned}\right\} \qquad (4)$$

The equations of motion for the transformed turbulence field are then

$$
\left.
\begin{aligned}
\hat{u}_t + A(\hat{u}\hat{u})_{\hat{x}} + B(\hat{v}\hat{u})_{\hat{y}} + C(\hat{w}\hat{u})_{\hat{z}} + \hat{p}_{\hat{x}} &= \nu(A\hat{u}_{\hat{x}\hat{x}} + B\hat{u}_{\hat{y}\hat{y}} + C\hat{u}_{\hat{z}\hat{z}}) \\
\hat{v}_t + A(\hat{u}\hat{v})_{\hat{x}} + B(\hat{v}\hat{v})_{\hat{y}} + C(\hat{w}\hat{v})_{\hat{z}} + \hat{p}_{\hat{y}} &= \nu(A\hat{v}_{\hat{x}\hat{x}} + B\hat{v}_{\hat{y}\hat{y}} + C\hat{v}_{\hat{z}\hat{z}}) \\
\hat{w}_t + A(\hat{u}\hat{w})_{\hat{x}} + B(\hat{v}\hat{w})_{\hat{y}} + C(\hat{w}\hat{w})_{\hat{z}} + \hat{p}_{\hat{z}} &= \nu(A\hat{w}_{\hat{x}\hat{x}} + B\hat{w}_{\hat{y}\hat{y}} + C\hat{w}_{\hat{z}\hat{z}}) \\
A\hat{u}_{\hat{x}} + B\hat{v}_{\hat{y}} + C\hat{w}_{\hat{z}} &= 0
\end{aligned}
\right\} \qquad (5)
$$

It is interesting to note that Taylor's theory (see ref. 2) for the sudden distortion of a turbulent field is particularly simple when expressed in these transformed variables, because strain *rate* does not appear. Suppose that a field of turbulence (state 1) is rapidly strained. The resulting field (state 2) is defined as

$$\hat{u}^{(2)} = \hat{u}^{(1)} + \int_0^\tau \hat{u}_t \, dt \ , \text{ etc.}$$

where τ is a time characteristic of the imposed straining period. Continuity must be satisfied throughout the straining process and, in particular, in the initial and final states of strain. Hence, if the total strain imposed is given by its inverses A,B,C, we have

$$\hat{u}^{(1)}_{\hat{x}} + \hat{v}^{(1)}_{\hat{y}} + \hat{w}^{(1)}_{\hat{z}} = 0 \qquad \text{at} \quad t$$

$$A\hat{u}^{(2)}_{\hat{x}} + B\hat{v}^{(2)}_{\hat{y}} + C\hat{w}^{(2)}_{\hat{z}} = 0 \qquad \text{at} \quad t + \tau$$

As the strain becomes sudden ($\tau \to 0$) A,B,C, $\hat{u}$, $\hat{v}$, and $\hat{w}$ remain bounded but their time derivatives do not. During the straining process the momentum equations (5) degenerate, as $\tau \to 0$, to

$$\hat{u}_t + \hat{p}_{\hat{x}} = 0$$

$$\hat{v}_t + \hat{p}_{\hat{y}} = 0$$

$$\hat{w}_t + \hat{p}_{\hat{z}} = 0$$

so that the velocity jump in the transformed variables is irrotational, that is

$$\hat{u}^{(2)} = \hat{u}^{(1)} - \int_0^\tau \hat{p}_{\hat{x}} \, dt = \hat{u}^{(1)} - \phi_{\hat{x}}$$

and similarly

$$\hat{v}^{(2)} = \hat{v}^{(1)} - \phi_{\hat{y}}$$
$$\hat{w}^{(2)} = \hat{w}^{(1)} - \phi_{\hat{z}}$$

The continuity condition then determines the potential as the solution of

$$A\phi_{\hat{x}\hat{x}} + B\phi_{\hat{y}\hat{y}} + C\phi_{\hat{z}\hat{z}} = A\hat{u}^{(1)}_{\hat{x}} + B\hat{v}^{(1)}_{\hat{y}} + C\hat{w}^{(1)}_{\hat{z}}$$

which completes the solution for the velocity field at state 2.

The above transformations seem to be the natural ones for the study of the effect of uniform imposed strain on a homogeneous turbulent field, regardless of the rate at which the strain is imposed. A more general set of transformations can be used when the mean strain-rate matrix is not diagonal, and also when the mean vorticity is nonzero.

NUMERICAL APPROXIMATIONS

We wish to simulate a spatially homogeneous turbulence field in an infinite space, and this suggests that we represent the field spatially as a Fourier series. The resulting field is periodic in all three space dimensions, with correspondingly periodic spatial correlations. However, if these correlations decay to negligible magnitude within the period, (e.g., if the integral scale is much smaller than half the period), the error due to the finite period should be small. In practice this requirement is difficult to satisfy with the resolution allowed by today's computers.

In this section we develop the equations in more detail and describe the integration process as programmed. Let $\tau_{11} = \hat{u}\hat{u}$, $\tau_{12} = \hat{u}\hat{v}$, $\tau_{13} = \hat{u}\hat{w}$, etc., tilde (~) denote the three-dimensional Fourier transform, and k_1, k_2, k_3 be wave numbers in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ directions, respectively. The equations (5) in wave space are then

$$\left.\begin{aligned}
\tilde{u}_t + ik_1A\tilde{\tau}_{11} + ik_2B\tilde{\tau}_{12} + ik_3C\tilde{\tau}_{13} + ik_1\tilde{p} &= -\nu(Ak_1^2 + Bk_2^2 + Ck_3^2)\tilde{u} \\
\tilde{v}_t + ik_1A\tilde{\tau}_{12} + ik_2B\tilde{\tau}_{22} + ik_3C\tilde{\tau}_{23} + ik_2\tilde{p} &= -\nu(Ak_1^2 + Bk_2^2 + Ck_3^2)\tilde{v} \\
\tilde{w}_t + ik_1A\tilde{\tau}_{13} + ik_2B\tilde{\tau}_{23} + ik_3C\tilde{\tau}_{33} + ik_3\tilde{p} &= -\nu(Ak_1^2 + Bk_2^2 + Ck_3^2)\tilde{w} \\
ik_1A\tilde{u} + ik_2B\tilde{v} + ik_3C\tilde{w} &= 0
\end{aligned}\right\} \quad (6)$$

The linear terms are combined by multiplying the equations by the integrating factor

$$F(\underline{k},t) = e^{\nu k_1^2 \int_0^t A\,dt}\; e^{\nu k_2^2 \int_0^t B\,dt}\; e^{\nu k_3^2 \int_0^t C\,dt} \tag{7}$$

giving

$$\begin{aligned}
&\frac{d}{dt}(F\tilde{u}) + F\{ik_1 A\tilde{\tau}_{11} + ik_2 B\tilde{\tau}_{12} + ik_3 C\tilde{\tau}_{13} + ik_1\tilde{p}\} = 0\\
&\frac{d}{dt}(F\tilde{v}) + F\{ik_1 A\tilde{\tau}_{12} + ik_2 B\tilde{\tau}_{22} + ik_3 C\tilde{\tau}_{23} + ik_2\tilde{p}\} = 0\\
&\frac{d}{dt}(F\tilde{w}) + F\{ik_1 A\tilde{\tau}_{13} + ik_2 B\tilde{\tau}_{23} + ik_3 C\tilde{\tau}_{33} + ik_3\tilde{p}\} = 0\\
&ik_1 A(F\tilde{u}) + ik_2 B(F\tilde{v}) + ik_3 C(F\tilde{w}) = 0
\end{aligned} \tag{8}$$

Now multiply the first equation by ik_1, the second by ik_2, etc., to obtain (let $\tilde{U} = ik_1\tilde{u}$, $\tilde{V} = ik_2\tilde{v}$, $\tilde{W} = ik_3\tilde{w}$)

$$\begin{aligned}
&\frac{d}{dt}(F\tilde{U}) = F\{k_1^2 A\tilde{\tau}_{11} + k_1 k_2 B\tilde{\tau}_{12} + k_1 k_3 C\tilde{\tau}_{13}\} + k_1^2 F\tilde{p}\\
&\frac{d}{dt}(F\tilde{V}) = F\{k_1 k_2 A\tilde{\tau}_{12} + k_2^2 B\tilde{\tau}_{22} + k_2 k_3 C\tilde{\tau}_{23}\} + k_2^2 F\tilde{p}\\
&\frac{d}{dt}(F\tilde{W}) = F\{k_1 k_3 A\tilde{\tau}_{13} + k_2 k_3 B\tilde{\tau}_{23} + k_3^2 C\tilde{\tau}_{33}\} + k_3^2 F\tilde{p}\\
&A(F\tilde{U}) + B(F\tilde{V}) + C(F\tilde{W}) = 0
\end{aligned} \tag{9}$$

(The purpose of this transformation of dependent variables is discussed later on; note that $k_1, k_2, k_3 = 0$ are special cases.)

The usual procedure for the computation of $\tilde{p}$ requires the time differentiation of the continuity condition. However, we want the algorithm to handle impulsive strains correctly (jumps in A, B, and C), that is, according to Taylor's sudden distortion theory, so we need to avoid the differentiation. We thus define a potential $\tilde{\phi}$ as

$$\tilde{\phi} = -F^{-1}\int_0^t F\tilde{p}\,dt$$

and absorb it into the time-advanced variables. Then

$$\left.\begin{aligned} \frac{d\tilde{X}}{dt} &= F\{k_1^2 A\tilde{\tau}_{11} + k_1 k_2 B\tilde{\tau}_{12} + k_1 k_3 C\tilde{\tau}_{13}\} \\ \frac{d\tilde{Y}}{dt} &= F\{k_1 k_2 A\tilde{\tau}_{12} + k_2^2 B\tilde{\tau}_{22} + k_2 k_3 C\tilde{\tau}_{23}\} \\ \frac{d\tilde{Z}}{dt} &= F\{k_1 k_3 A\tilde{\tau}_{13} + k_2 k_3 B\tilde{\tau}_{23} + k_3^2 C\tilde{\tau}_{33}\} \end{aligned}\right\} \qquad (10)$$

and the continuity condition becomes

$$\tilde{\phi} = F^{-1}\left(\frac{A\tilde{X} + B\tilde{Y} + C\tilde{Z}}{Ak_1^2 + Bk_2^2 + Ck_3^2}\right) \qquad (11)$$

where

$$\left.\begin{aligned} F^{-1}\tilde{X} &= \tilde{U} + k_1^2\tilde{\phi} \\ F^{-1}\tilde{Y} &= \tilde{V} + k_2^2\tilde{\phi} \\ F^{-1}\tilde{Z} &= \tilde{W} + k_3^2\tilde{\phi} \end{aligned}\right\} \qquad (12)$$

The $\tilde{\tau}$'s are functions of $\tilde{u}$, $\tilde{v}$, $\tilde{w}$ only, so that, if $\tilde{u}$, $\tilde{v}$, $\tilde{w}$ are known at the beginning of a time step and satisfy the continuity condition, we may advance $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$. However, to form time derivatives of the advanced $\tilde{X}$, $\tilde{Y}$, $\tilde{Z}$ we must form from (12) advanced values of $\tilde{u}$, $\tilde{v}$, $\tilde{w}$, and this requires the solution (11) for $\tilde{\phi}$ *at the advanced time*. This is done using the continuity condition at the advanced time, and does not require its time differentiation.

At the beginning of a time step $t = 0$, and we have

$$F = 1 \ , \ \tilde{\phi} = 0 \ , \ \tilde{X} = \tilde{U} \ , \ \tilde{Y} = \tilde{V} \ , \ \tilde{Z} = \tilde{W}$$

The equations for $\tilde{X}$, $\tilde{Y}$, and $\tilde{Z}$ are integrated over the time step, and the final values are used in equations (11) and (12) to produce final values of $\tilde{U}$, $\tilde{V}$, and $\tilde{W}$. The origin of time is then shifted to the final time giving the proper initialization for the next time step.

Spatial differentiation is a point operator in wave space but multiplication (e.g., $\tilde{\tau}_{12} = \widetilde{\hat{u}\hat{v}}$) is not, and the most efficient means of forming the Fourier transform of a product from the transforms of its terms is to return to physical space by inverting the transforms, form the product, and then transform the result back to wave space. Unfortunately, the transformation of the product back to wave space introduces an error due to spectral truncation.

The truncation errors are most easily demonstrated in one spatial dimension. The representation of the product of two Fourier series $\tilde{a}$, $\tilde{b}$ (in complex form) as a Fourier series $\tilde{c}$ is given by the (infinite) convolution sum

$$\tilde{c}_k = \sum_{s=-\infty}^{+\infty} \tilde{a}_{k-s}\tilde{b}_s$$

However, the process of inverting *finite* transforms $\tilde{a}$ and $\tilde{b}$, forming the product ab, and then taking its *finite* transform results instead in two sums:

$$\tilde{c}_k = \sum_s \tilde{a}_{k-s}\tilde{b}_s + \sum_s \tilde{a}_{k\pm M-s}\tilde{b}_s$$

The first sum represents a contribution (incomplete due to truncation) to ab correctly attributed to wave number k. The second sum also represents a contribution to ab, but it is actually a contribution not to k, but to $k\pm M$, wave numbers beyond those allowed by the length (M) of the finite transforms used. This is the "aliasing" error. Now it may be argued that because aliasing errors do not account for all of the truncation error, suppression of the aliasing error is not cost effective so far as accuracy is concerned. However, in the algorithm used here, the aliased terms can lead to nonlinear instability, and their control is essential.

Now consider the effect of a shift of the physical coordinate system. In wave space this amounts to multiplication by $e^{ik\Delta}$, where $-\Delta$ is the amount of coordinate shift. If we use $e^{ik\Delta}$ to shift $\tilde{a}_k$, $\tilde{b}_k$ prior to inverting them to physical space, form the product ab on the shifted grid, transform back to wave space, and finally shift coordinates back with $e^{-ik\Delta}$ we obtain

$$\tilde{c}_k = \sum_s \tilde{a}_{k-s}\tilde{b}_s + e^{\pm iM\Delta} \sum_s \tilde{a}_{k\pm M-s}\tilde{b}_s$$

The first (alias-free) sum is invariant under these shifts, but the second sum, the aliased one which we wish to suppress, has a phase dependency on Δ and can be eliminated. For example, if two evaluations are made, one with $e^{\pm iM\Delta} = 1$ and the other with $e^{\pm iM\Delta} = -1$, the alias-free result is one-half their sum. The second sum (which is multiplied by the phase factor) itself vanishes identically for $|k| \le N$, $(N < M/3)$ if modes of $\tilde{a}$ and $\tilde{b}$ outside of this range are nulled prior to inversion, and transforms of length M are retained. Thus two independent procedures are available for alias suppression.

The extension of these procedures to three dimensions gives for each $\tilde{c}_{\underline{k}}$ eight terms, seven of which represent aliasing errors. The aliased terms are classified according to the number of dimensions in which aliasing has occurred. We then have [$\underline{k} = (k_1, k_2, k_3)$, $\theta_n = e^{\pm ik_n\Delta_n}$]

$$
\begin{aligned}
c_{\underline{k}} &= S_0 && \text{(alias-free)} \\
&+ \theta_1 S_1 + \theta_2 S_2 + \theta_3 S_3 && \text{(singly-aliased)} \\
&+ \theta_1\theta_2 S_4 + \theta_2\theta_3 S_5 + \theta_3\theta_1 S_6 && \text{(doubly-aliased)} \\
&+ \theta_1\theta_2\theta_3 S_7 && \text{(triply-aliased)}
\end{aligned}
$$

All of the aliased sums (S_1, . . . S_7) vanish if modes having *any* $k_i > N_i$ are nulled. The doubly and triply aliased sums (S_4, . . ., S_7) vanish if modes having *any two* $k_i > N_i$ are nulled. The triply aliased sum (S_7) vanishes if modes having *all three* $k_i > N_i$ are nulled. Alternatively one can evaluate the convolution eight times using the eight combinations of θ_x, θ_y, $\theta_z = \pm 1$ and sum to eliminate the aliased terms. Note that suppression by the latter means requires eight evaluations to eliminate all of the aliased terms. One can also, as suggested by Orszag (ref. 1), remove S_4, . . . S_7 by truncation and the remaining single aliases by coordinate shift with two evaluations. We are faced with the choice between losing information (truncation) or losing computational speed (multiple evaluations).

We have, following Orszag, eliminated doubly and triply aliased sums by truncation, though the truncation used here differs slightly from that of Orszag who nulls modes having $\underline{k} \cdot \underline{k} > 2(M/3)^2$. We have not exactly eliminated the remaining single aliases, due to the computational cost of the double evaluations required. Instead we have used the fact that the Runge-Kutta algorithm requires pairs of evaluations at each half step and that by using a shifted grid for the second evaluation we reduce the total alias error for the pair by a factor of Δt^2. The possibility of nonlinear instability is further reduced by insuring that the θ_j for the first evaluation in a pair are not correlated with those of other pairs. This is easily accomplished by the use of a uniform-random-number generator during computation of the phase factors.

DATA MANAGEMENT

In large simulations the high-speed random-access memory of the computer cannot hold the entire data base of the problem (in the present code it holds 6% of it). In this case the high-speed memory may only be able to hold a few lines of the mesh (e.g., all values of k_1 for a few k_2, k_3 values), and it is convenient to transform and take derivatives only along those lines. In general, separate passes over the data base are required for each spatial dimension. The directional order in which operations are performed then determines the required number of passes over the data base. We will demonstrate how this number may be reduced in a spectral algorithm.

Consider the evaluation in wave space of $(\hat{u}\hat{v})_{\hat{x}}$ and $(\hat{u}\hat{v})_{\hat{y}}$, which is required in equation (5). The transforms of $\hat{u}$ and $\hat{v}$ are inverted in the $\hat{x}$, $\hat{y}$, and $\hat{z}$ directions, each direction requiring a separate pass over the

data base. On the last ($\hat{z}$) pass of this sequence we also form, in physical space, the $\hat{u}\hat{v}$ product and then transform back to wave space in the $\hat{z}$ direction. In principle there remain only the $\hat{x}$ and $\hat{y}$ transforms and the multiplications by ik_x and ik_y to form the derivatives in the $\hat{x}$ and $\hat{y}$ directions. The problem is that, under our constraints, transforms and derivatives can only be taken in the direction of the grid lines held in fast memory. Under these constraints we must either perform three transforms and two derivatives in two passes, or two transforms and two derivatives in three passes. If the constraint on the derivative is absent the results can be obtained in two transforms and two derivatives in two passes. This constraint can be removed only if four lines of the mesh can be held simultaneously in fast memory (so that all eight real numbers representing wave number $\underline{k}$ are present). The ILLIAC fast memory is sufficiently large to accommodate four mesh lines, but not within a single processing element (PE), so that differentiation would require communication across the PE's. We have instead used a slightly altered set of dependent variables that avoids this problem altogether.

If the $\hat{x}$ momentum equation is differentiated with respect to $\hat{x}$, and the $\hat{y}$ momentum equation with respect to $\hat{y}$, the $\hat{u}\hat{v}$ stress term appears as $(\hat{u}\hat{v})_{\hat{x}\hat{y}}$ in both equations, and its evaluation under the constraints requires two transforms and two derivatives in two passes. But two extra integrations (of $\hat{u}_{\hat{x}}$ and $\hat{v}_{\hat{y}}$) are then required to form $\hat{u}$ and $\hat{v}$ in physical space; however, since integration and differentiation cost far less than either a transform or an I/O pass, this method is quite efficient. To avoid loss, upon differentiation, of information in a Fourier mode having a null wave number we simply do not multiply that mode by its wave number (i.e., zero) and similarly when we integrate it we do not divide by its wave number. What this amounts to is that, instead of the usual spectral dependent variables

$$\tilde{u}(k_1,k_2,k_3)$$

$$\tilde{v}(k_1,k_2,k_3)$$

$$\tilde{w}(k_1,k_2,k_3)$$

we use

$$\tilde{u}(0,k_2,k_3)\ ,\quad ik_1\tilde{u}(k_1,k_2,k_3)\ ,\quad k_1 \neq 0$$

$$\tilde{v}(k_1,0,k_3)\ ,\quad ik_2\tilde{v}(k_1,k_2,k_3)\ ,\quad k_2 \neq 0$$

$$\tilde{w}(k_1,k_2,0)\ ,\quad ik_3\tilde{w}(k_1,k_2,k_3)\ ,\quad k_3 \neq 0$$

Use of these variables simplifies the continuity condition and minimizes the number of transforms and passes over the data base.

INITIAL CONDITIONS

The initial velocities are chosen randomly, subject to the constraints of continuity and a specified energy spectrum. In detail, the real and imaginary parts of the Fourier velocity amplitudes $\underline{u}(\underline{k})$ are selected randomly from a uniform distribution over the circle that is the intersection of the sphere (having surface area proportional to $E(k)$) determined by the desired energy spectrum and the plane (normal to $\underline{k}$) determined by continuity. For example, consider the (real) spectral mode ($k_1, k_2, k_3 \neq 0$)

$$u_x = f_1 \cos k_1 x \sin k_2 y \cos k_3 z$$

$$v_y = f_2 \cos k_1 x \sin k_2 y \cos k_3 z$$

$$w_z = f_3 \cos k_1 x \sin k_2 y \cos k_3 z$$

The algorithm described previously advances the vector $\underline{f}$ in time given the initial values ($k_1, k_2, k_3 = 0$ are special cases)

$$f_1 = c(k_1^2 + k_2^2)^{-1/2} k_1 \left(k_2 \cos \psi + \frac{k_1 k_3}{k} \sin \psi \right)$$

$$f_2 = c(k_1^2 + k_2^2)^{-1/2} k_2 \left(\frac{k_2 k_3}{k} \sin \psi - k_1 \cos \psi \right)$$

$$f_3 = -c(k_1^2 + k_2^2)^{1/2} \frac{k_3}{k} \sin \psi$$

where $k^2 = k_1^2 + k_2^2 + k_3^2$, $c^2 \approx [E(k)/2\pi k^2]$, and ψ is a random number uniformly distributed on the interval $(0, 2\pi)$.

SCALING PROCEDURE

The simulation variables are nondimensional (integer wave numbers with period 2π) and must be scaled to obtain dimensional values. If we wish to simulate an experimental flow, knowing at the initial time only its energy spectrum $E_e(k_e)$ and viscosity ν_e, we must use a similar energy spectrum (i.e., differing only in energy and wave number scales) and specify a simulation viscosity such that the dimensionless problems are the same. Thus we define scale factors α, β relating the simulation energy E and wave number k to the experimental values by

$$E(k)dk = \alpha E_e(k_e)dk_e \ , \quad k_e = \beta k$$

Here E and k are dimensionless, α has units $L^{-2}T^2$, and β has units L^{-1} so that time must scale as

$$t = \alpha^{-1/2}\beta t_e$$

and the viscosity as

$$\nu = \alpha^{1/2}\beta\nu_e$$

The scale factor α depends only on the scaling of the dependent variables of the problem and, since the computation allows an unlimited range for their values, changes of α produce only changes of scale in the results. This would also be the case for β if the computation allowed an unlimited range of values for the independent variables (wave numbers), but such would require infinite spatial resolution.

The computer simulation does not, of course, have infinite resolution, and the range of its independent variables is simply the value of its highest nondimensional (integer) wave number. Because the entire experimental range of length scales cannot be simulated, the choice of β determines *which* physical wave numbers of the experiment are to be simulated. Clearly the error of the simulation depends on this choice, but at present no rationale is known for making it. The choice of Ferziger (ref. 3), for example, is the scaling that allows the greatest total energy to be included in the simulation. There is also, however, the implicit constraint that the spatial period be "much greater" than the integral scale of the turbulent field.

The number of modes, or degree of freedom of the motion, within a *range* of scales (e.g., $K < k < 2K$) characterized by K is proportional to K^3. The different scale ranges of the motion are certainly not represented equally well in the statistical sense. There are very few modes in the larger scales (smaller wave numbers), and if the simulation is to represent turbulence we must require that a small number of modes does not contain a large fraction of the energy. This is equivalent to the integral length constraint.

If the truncation error is to be small, the viscosity must be high enough to damp the highest wave numbers of the calculation to the point where they, and presumably also those lost by truncation, have negligible effect. In other words, an accurate and complete solution requires that the computation resolve all scales of motion; otherwise one must face the notorious "closure problem." Since the simulation code presented here contains no closure approximations, it is necessarily restricted to very low-turbulence Reynolds numbers.

SAMPLE RESULTS

Several runs of the simulation code have been made for isotropic flow to develop the algorithm and to obtain numerical error estimates. Typical energy spectra are shown in figures 3-6. The algorithm used has negligible numerical dissipation and, when the spatial resolution is truncated at any

finite wave number in an inviscid ($\nu = 0$) fluid, the energy cascade eventually leads to an equipartitioned state (fig. 3) with the theoretical spectrum $E(k) = ck^2$. This tendency is still apparent when molecular dissipation is included (fig. 4) if the range of computed scales does not include almost all of the dissipation. More of the dissipation range can be included by increasing the range of computed scales (fig. 5) or by shifting the range of computed scales (fig. 6) to include more of the dissipation range.

APPENDIX

THE ILLIAC PROGRAM

Program Structure

A fourth-order Runge-Kutta algorithm is used to integrate the system of equations (10)-(12). The strain inverses A, B, C, and the integrating factor F are considered known. The bulk of the computation is the evaluation of the right side of (10), which is done in subroutines PHASE1, PHASE2, and PHASE3. The dependent variables $\tilde{X},\tilde{Y},\tilde{Z}$, are then advanced in STEP, and the continuity condition (11), is used by PRESSR to recover the physical velocities (12). These five subprograms are called sequentially by the control routine LOOP which is responsible for data management and step control.

The functions of processes called by these routines are given by in-line comments in the listing.

Data Structure and Flow

The data base resides on disk and consists of two blocks. The first block of data holds the velocity field at the beginning of a Runge-Kutta step (three words/node) and a predicted velocity accumulator field (three words/node). This block of data is always accessed sequentially. The second block of data is working space (four words/node) in which the right side of (10) is evaluated, requiring both sequential and nonsequential page accesses from the disk.

Each prediction within the Runge-Kutta process requires two complete passes through the data base, one bringing (x,y) planes into core (PHASE1, PHASE3, STEP, and PRESSR) for operators in the y direction, and one bringing in (x,z) planes (PHASE2) for operators in the x and z directions. In the latter pass, only the working space data block is required, allowing the (x,z) planes to be handled by a triple buffered scheme.

Listing of Program

The program is coded for execution in 32-bit precision on the ILLIAC computer. The routines listed in this appendix, which are coded in the CFD language, cover the major algorithmic steps of the computation. Some of the lower level routines are coded in assembly language (ASK) for efficiency, and others had to be hand coded because of the restrictions placed on 32-bit operation by the CFD language.

COMPUTER ROUTINES

SUBROUTINE LOOP

```
C*************************************************************************
C*     MAIN LOOP-DRIVING PROGRAM FOR SOLUTION OF 3-D NAVIER-STOKES EQNS   *
C*************************************************************************
      *SUBROUTINE LOOP
      *EXTERNAL PHASE1, PHASE2, DEFORM, STEP, FFIELD,                -
      *                 RCOL, WCOL, RROW, WROW, SPCTRA, RSTART, RSHIFT
      *CU INTEGER PLANE, K, RKSTEP, MAXSTP, NSTEP, Z, KMAX, EPAGE,   -
      *           LAST, OUTPUT, SKEW1, OFFSET, BASE1, BASE2, FSTEP
      *CU REAL CFL, DELTAT, A, B, AB
      *EQUIVALENCE (9,PLANE), (10,K), (4,RKSTEP), (8,MAXSTP), (7,NSTEP),-
      *            (2,Z), (17,KMAX), (18,EPAGE), (28,CFL), (3,DELTAT),  -
      *            (21,SKEW1), (22,OFFSET), (29,BASE1), (30,BASE2),     -
      *            (14,LAST), (15,OUTPUT), (32,FSTEP), (11,A), (12,B),  -
      *            (13,AB)
      *PE INTEGER XYPLAN, SSUB(5)
      *DATA SSUB /1, 2, 2, 3, 3/
      *COMMON /STRAIN/ TIME(*), SCALES(*,3,3)
      *COMMON /XYORDR/ XYPLAN(*)
      *COMMON /PASS1/ P1(*,640,2)
      *COMMON /PASS2/ P2(*,512,3)
      *COMMON /KSPACE/ RK(*), RKSQR(*), CHOP(*), VMAX(*)
C*****************************   INITIALIZE POINTERS AND COUNTERS   **************
       LAST = 0                                          '  LAST PASS FLAG
       EPAGE = 0                                         '  OUTPUT PAGE COUNTER
       NSTEP = 0                                         '  STEPS COMPLETED COUNTER
      *CALL DEFORM                                       '  INITIALIZE MEAN STRAINS
      *CALL RSTART                                       '  SEED RANDOM NUM. GENERATOR
      *CALL RSHIFT                                       '  INITIALIZE COORDINATE SHIFTS
       RKSTEP = 0                                        '  RUNGE-KUTTA SUB-STEP COUNTER
       BASE1 = 0                                         '  DISK AREA OFFSETS
       BASE2 = SKEW1
       OFFSET = 0
      *GO TO 4
C*************************************************************   MAIN LOOP   *********
   10 NSTEP = NSTEP + 1                                  '  ANOTHER STEP IN THE BAG
      RKSTEP = 1                                         '  RE-SET RUNGE-KUTTA STEP #
     *CALL RSHIFT                                        '  GENERATE COORDINATE SHIFTS
    8 VMAX(*) = CFL / (FLOAT(KMAX) *                    -'  NEXT TIME STEP
     *                 ROWMAX(VMAX(*)))
```

SUBROUTINE LOOP (CONT.)

```
      DELTAT = VMAX(1)
      VMAX(*) = 0.
     *CALL DEFORM                                    '  APPLY MEAN STRAIN
C*******************     PASS 1 THRU DATA BASE  (IN Z DIRECTION)     ***********
    4*CALL RCOL(1, P1(*,1,1), XYPLAN(1))             '  PRE-LOAD BUFFERS
     *CALL RCOL(2, P1(*,1,2), XYPLAN(3))
      K = SSUB(RKSTEP+1)
      A = SCALES(3,1,K)
      B = SCALES(3,2,K)
      AB = SCALES(3,3,K)
      K = 1                                          '  BUFFER POINTER
     *DO 1 PLANE = 1, 64, 2                          '  LOOP OVER ALL Z-PLANES
      Z = XYPLAN(PLANE)
     *WAIT K, K+2                                    '  WAIT ON BUFFER FLUSH / LOAD
     *CALL STEP(P1(*,1,K))                           '  RUNGE-KUTTA STEP ON VELOCITY
     *IF(LAST .EQ. 0) CALL PHASE1(P1(*,1,K))         '  BEGIN TIME DERIVATIVES
     *CALL WCOL(K+2, P1(*,1,K), XYPLAN(PLANE))       '  FLUSH BUFFER
     *IF(PLANE + 4 .GT. 64) GO TO 1
     *CALL RCOL(K, P1(*,1,K), XYPLAN(PLANE+4))       '  RELOAD BUFFER
    1 K = 3 - K                                      '  NEXT BUFFER
C*********************************     END OF PASS 1     ***********************
     *IF(LAST .NE. 0) GO TO 6                         '  TEST FOR END OF RUN
      OFFSET = 1 - OFFSET                            '  SWAP DISK AREAS
      BASE1 = BASE2                                  '  ORIGIN OF SOURCE AREA
      BASE2 = BASE1 + SKEW1                          '  ORIGIN OF DESTINATION AREA
     *IF(BASE2 .GT. 299) BASE2 = BASE2 - 300
C*******************     PASS 2 THRU DATA BASE  (IN Y DIRECTION)     ***********
     *CALL RROW(1, P2(*,1,1), 1)                     '  PRE-LOAD BUFFERS
     *CALL RROW(2, P2(*,1,2), 5)
     *CALL RROW(3, P2(*,1,3), 9)
      K = 1                                          '  BUFFER POINTER
     *DO 2 PLANE = 1, 64, 4                          '  LOOP OVER Y-PLANES (4 PER)
     *WAIT K, K+3
     *CALL PHASE2(P2(*,1,K))                         '  CONTINUE TIME DERIVATIVES
     *CALL PHASE2(P2(*,5,K))
     *CALL WROW(K+3, P2(*,1,K), PLANE)               '  FLUSH / LOAD BUFFER
     *IF(PLANE + 12 .GT. 64) GO TO 102
     *CALL RROW(K, P2(*,1,K), PLANE+12)
  102 K = K + 1                                      '  NEXT BUFFER
```

```
    2*IF(K .GT. 3) K = 1
C**********************************      END OF PASS 2     ************************
      RKSTEP = RKSTEP + 1                          '  NEXT RUNGE-KUTTA SUBSTEP
     *GO TO (8, 4, 4, 9, 10), RKSTEP
    9*IF(NSTEP + 1 .EQ. MAXSTP) LAST = 1           '  TERMINATE ON STEP LIMIT
     *IF(NSTEP .EQ. FSTEP) CALL FFIELD
     *GO TO 4
C**************************************      END OF MAIN LOOP     *****************
    6*WAIT                                         '  COMPUTATION TERMINATED
   11*DO 12 Z = 1, 1600                            '  SQUASH DUMP FILE
   12 P1(*,Z,1) = 0.
     *RETURN
     *END
```

SUBROUTINE STEP

```
C*****************************************************************************
C*      RUNGE-KUTTA INTEGRATION PROCESS                                      *
C*****************************************************************************
      *SUBROUTINE STEP(BUFFER)
      *COMMON /BUFFER/ U(*,4,64), UN(*,64), VN(*,64), WN(*,64),          -
      *                            DU(*,64), DV(*,64), DW(*,64)
      *EXTERNAL PHASE3, PRESSR, TRUNC, SPCTRA
      *CU INTEGER I, RKSTEP, Z
      *EQUIVALENCE (1,I), (4,RKSTEP), (2,Z)
      *COMMON /KSPACE/ RK(*), RKSQR(*), CHOP(*)
      *COMMON /KSPLIT/ RKZ(*), RKZSQR(*), CHOPZ(*)
      *COMMON /SCRTCH/ FPART(*), FACTOR(*)
      *COMMON /STRAIN/ TIME(*), SCALES(*,3,3), VCHOP(*,3,4)
      *COMMON /ABC1/ UNITY(*,3)
      *GO TO (1, 2, 3, 4), RKSTEP
C*************************   SPECIAL STARTING STEP   **************************
       FPART(*) = CHOPZ(Z) * CHOP(*)
      *DO 100 I = 1, 64
       FACTOR(*) = CHOP(I) * FPART(*)
      *CALL TRUNC(FACTOR)                          '  TRUNCATE 2-D, 3-D ALIASES
       UN(*,I) = FACTOR(*) * UN(*,I)               '  SHARP FILTER DATA
       VN(*,I) = FACTOR(*) * VN(*,I)
       WN(*,I) = FACTOR(*) * WN(*,I)
       U(*,1,I) = UN(*,I)                          '  STARTING FIELD
       U(*,2,I) = VN(*,I)
  100  U(*,3,I) = WN(*,I)
      *CALL PRESSR(U, SCALES(*,1,1))               '  ENFORCE CONTINUITY @ START
      *CALL SPCTRA(BUFFER)                         '  SPECTRA OF INITIAL FIELD
      *RETURN
C*************************   RUNGE-KUTTA STEP 1   *****************************
     1*CALL PHASE3(U, SCALES(*,1,1), UNITY)        '  INERTIAL RATE @ 0
       FPART(*) = VCHOP(*,1,2) * VCHOP(Z,3,2)
      *DO 10 I = 1, 64
       FACTOR(*) = FPART(*) * VCHOP(I,2,2)         '  INVERSE FILTER @ H/2
      *CALL TRUNC(FACTOR)                          '  TRUNCATE 2-D, 3-D ALIASES
       DU(*,I) = U(*,1,I)                          '  ACCUMULATE SUBSTEP
       DV(*,I) = U(*,2,I)
       DW(*,I) = U(*,3,I)
       U(*,1,I) = FACTOR(*) * (UN(*,I) + U(*,1,I) / 2.)    '  PREDICT @ H/2
```

```
      U(*,2,I) = FACTOR(*) * (VN(*,I) + U(*,2,I) / 2.)
   10 U(*,3,I) = FACTOR(*) * (WN(*,I) + U(*,3,I) / 2.)
     *CALL PRESSR(U, SCALES(*,1,2))                    '  ENFORCE CONTINUITY @ H/2
     *RETURN
C*****************************   RUNGE-KUTTA STEP 2   ******************************
    2*CALL PHASE3(U, SCALES(*,1,2), VCHOP(*,1,1))      '  INERTIAL RATE @ H/2
      FPART(*) = VCHOP(*,1,2) * VCHOP(Z,3,2)
     *DO 20 I = 1, 64
      FACTOR(*) = FPART(*) * VCHOP(I,2,2)              '  INVERSE FILTER @ H/2
     *CALL TRUNC(FACTOR)                               '  TRUNCATE 2-D, 3-D ALIASES
      DU(*,I) = DU(*,I) + U(*,1,I) * 2.                '  ACCUMULATE SUBSTEP
      DV(*,I) = DV(*,I) + U(*,2,I) * 2.
      DW(*,I) = DW(*,I) + U(*,3,I) * 2.
      U(*,1,I) = FACTOR(*) * (UN(*,I) + U(*,1,I) / 2.)     '  PREDICT @ H/2
      U(*,2,I) = FACTOR(*) * (VN(*,I) + U(*,2,I) / 2.)
   20 U(*,3,I) = FACTOR(*) * (WN(*,I) + U(*,3,I) / 2.)
     *CALL PRESSR(U, SCALES(*,1,2))                    '  ENFORCE CONTINUITY @ H/2
     *RETURN
C*****************************   RUNGE-KUTTA STEP 3   ******************************
    3*CALL PHASE3(U, SCALES(*,1,2), VCHOP(*,1,1))      '  INERTIAL RATES @ H/2
      FPART(*) = VCHOP(*,1,4) * VCHOP(Z,3,4)
     *DO 30 I = 1, 64
      FACTOR(*) = FPART(*) * VCHOP(I,2,4)              '  INVERSE FILTER @ H
     *CALL TRUNC(FACTOR)                               '  TRUNCATE 2-D, 3-D ALIASES
      DU(*,I) = DU(*,I) + U(*,1,I) * 2.                '  ACCUMULATE SUBSTEP
      DV(*,I) = DV(*,I) + U(*,2,I) * 2.
      DW(*,I) = DW(*,I) + U(*,3,I) * 2.
      U(*,1,I) = FACTOR(*) * (UN(*,I) + U(*,1,I))          '  PREDICT @ H
      U(*,2,I) = FACTOR(*) * (VN(*,I) + U(*,2,I))
   30 U(*,3,I) = FACTOR(*) * (WN(*,I) + U(*,3,I))
     *CALL PRESSR(U, SCALES(*,1,3))                    '  ENFORCE CONTINUITY @ H
     *RETURN
C*****************************   RUNGE-KUTTA STEP 4   ******************************
    4*CALL PHASE3(U, SCALES(*,1,3), VCHOP(*,1,3))      '  INERTIAL RATES @ H
      FPART(*) = VCHOP(*,1,4) * VCHOP(Z,3,4)
     *DO 40 I = 1, 64
      FACTOR(*) = FPART(*) * VCHOP(I,2,4)              '  INVERSE FILTER @ H
     *CALL TRUNC(FACTOR)                               '  TRUNCATE 2-D, 3-D ALIASES
```

SUBROUTINE STEP (CONCL.)

```
      U(*,1,I) = FACTOR(*) * (UN(*,I) + (DU(*,I) +          -'  CORRECT @ H
     *                        U(*,1,I)) / 6.)
      U(*,2,I) = FACTOR(*) * (VN(*,I) + (DV(*,I) +          -
     *                        U(*,2,I)) / 6.)
40    U(*,3,I) = FACTOR(*) * (WN(*,I) +(DW(*,I) +           -
     *                        U(*,3,I)) / 6.)
     *CALL PRESSR(U, SCALES(*,1,3))                        '  ENFORCE CONTINUITY @ H
     *CALL SPCTRA(BUFFER)                                  '  SPECTRA OF FIELD
     *DO 41 I = 1, 64
      UN(*,I) = U(*,1,I)                                   '  UN-FILTERED FINAL FIELD
      VN(*,I) = U(*,2,I)                                   '  BECOMES FILTERED INITIAL
41    WN(*,I) = U(*,3,I)                                   '  FIELD FOR NEXT STEP
     *RETURN
     *END
```

SUBROUTINE PHASE1

```
C*****************************************************************************
C*     STARTING WITH TRANSFORMS OF THE VELOCITY DERIVATIVES DU/DX, DV/DY,     *
C*     DW/DZ PERFORM INTEGRATIONS AND INVERSE TRANSFORMS FOR RETURN, IN       *
C*     PHYSICAL SPACE, OF U, V, W.  COMPUTATION IN Y DIRECTION                *
C*****************************************************************************
      *SUBROUTINE PHASE1(P)
      *DIMENSION P(*,256)
      *EXTERNAL IFFT, IDIFF, SHIFT1
      *CU INTEGER I, LSKIP, RKSTEP
      *EQUIVALENCE (1,I), (24,LSKIP), (4,RKSTEP)
      *COMMON /ALIAS/ RNXYZ(*,3), XSHIFT(*,5), YSHIFT(*,5), ZSHIFT(*,5)
       LSKIP = 4
      *CALL SHIFT1(P(*,1), YSHIFT(*,RKSTEP+1))' RANDOM Y-COORDINATE SHIFT
      *CALL IDIFF(P(*,2))                     '  INTEGRATE DV/DY TO V
      *DO 1 I = 1, 3
     1*CALL IFFT(P(*,I))                      '  INVERT Y-TRANSFORM OF DU/DX, V, DW/DZ
      *RETURN
      *END
```

SUBROUTINE PHASE2

```
C*************************************************************************
C*     COMPLETE, IN (X,Z) PLANES INTEGRATION AND INVERSE TRANSFORMS TO OBTAIN  *
C*     U, V, W IN PHYSICAL SPACE.  FORM VELOCITY PRODUCTS (REYNOLDS STRESSES)  *
C*     AND TRANSFORM AND DIFFERENTIATE THEM IN THE X AND Z DIRECTIONS.         *
C*************************************************************************
      *SUBROUTINE PHASE2(P)
      *DIMENSION P(*,8,64)
      *EXTERNAL FFT, IFFT, DIFF, IDIFF, SHIFT1, SHIFT2, TRANS, GETMAX,  -
      *         WSWAP, SHIFT3
      *CU REAL AX, BY, ABZ
      *CU INTEGER I, LSKIP, RKSTEP
      *EQUIVALENCE (1,I), (24,LSKIP), (4,RKSTEP), (11,AX), (12,BY),     -
      *            (13,ABZ)
      *COMMON /PASS2/ P2(*,512,3), UW(*,64)
      *COMMON /KSPACE/ RK(*), RKSQR(*), CHOP(*), VMAX(*), FACT01(*)
      *COMMON /ALIAS/ RNXYZ(*,3), XSHIFT(*,5), YSHIFT(*,5), ZSHIFT(*,5)
      *COMMON /SCRTCH/ BV2(*), A(*), AB(*)
       LSKIP = 8
      *DO 6 I = 1, 3                        '  RESTRUCTURE DATA FOR PASS 2
     6*CALL WSWAP(P(*,I,1))
      *CALL SHIFT1(P, ZSHIFT(*,RKSTEP+1))   '  RANDOM Z-COORDINATE SHIFT
      *CALL IDIFF(P(*,3,1))                 '  INTEGRATE DW/DZ TO W
      *DO 1 I = 1, 3
      *CALL IFFT(P(*,I,1))                  '  INVERT Z-TRANSFORM OF DU/DX, V, W
     1*CALL TRANS(P(*,I,1))                 '  ALLIGN X DOWN PEM
      *CALL SHIFT1(P, XSHIFT(*,RKSTEP+1))   '  RANDOM X-COORDINATE SHIFT
      *CALL IDIFF(P(*,1,1))                 '  INTEGRATE DU/UX TO U
      *DO 7 I = 1, 3
     7*CALL IFFT(P(*,I,1))                  '  INVERT X-TRANSFORM OF U, V, W
C*****************************************      WELCOME TO PHYSICAL SPACE
      *GO TO (4, 4, 4, 5), RKSTEP
     5*CALL GETMAX(P, VMAX)                 '  FIND MAX VELOCITY (TO INSURE ACCURACY)
     4*DO 2 I = 1, 64
       BV2(*) = BY * P(*,2,I) ** 2                              '  BVV
       UW(*,I) = P(*,1,I) * P(*,3,I) / 262144.                  '  UW
       P(*,4,I) = P(*,2,I) * P(*,3,I) / 262144.                 '  VW
       P(*,3,I) = (ABZ * P(*,3,I) ** 2 - BV2(*)) / 262144.      '  CWW - BVV
       P(*,2,I) = P(*,1,I) * P(*,2,I) / 262144.                 '  UV
     2 P(*,1,I) = (AX * P(*,1,I) ** 2 - BV2(*)) / 262144.       '  AUU - BVV
```

SUBROUTINE PHASE2 (CONCL.)

```
C*********************************************     LEAVING PHYSICAL SPACE
     *DO 3 I = 1, 4                              '  FOR AUU-BVV, UV, CWW-BVV, VW
    3*CALL FFT(P(*,I,1))                         '  X TRANSFORM
     *CALL SHIFT2(P, XSHIFT(*,RKSTEP+1))         '  SHIFT X BACK
     *CALL DIFF(P(*,2,1))                        '  D(UV)/DX
     *DO 8 I = 1, 4                              '  FOR AUU-BVV, UV, CWW-BVV, VW
     *CALL TRANS(P(*,I,1))                       '  ALLIGN Z DOWN PEM
    8*CALL FFT(P(*,I,1))                         '  Z TRANSFORM
     *CALL DIFF(P(*,4,1))                        '  D(VW)/DZ
      LSKIP = 1                                  '  DO FOR THE UW PRODUCT .....
     *CALL FFT(UW)                               '  X TRANSFORM
     *CALL SHIFT3(UW, XSHIFT(*,RKSTEP+1))        '  SHIFT X BACK
     *CALL DIFF(UW)                              '  D(UW)/DX
     *CALL TRANS(UW)                             '  ALLIGN Z DOWN PEM
     *CALL FFT(UW)                               '  Z TRANSFORM
     *CALL DIFF(UW)                              '  D(UW)/DXDZ
      LSKIP = 8
      A(*) = AX * FACT01(*)
      AB(*) = ABZ * FACT01(*)                    '  BECAUSE DATA STRUCTURE NOT AS PASS 1
     *DO 10 I = 1, 64                            '  COMBINE STRESSES
      P(*,1,I) = RKSQR(*) * P(*,1,I) - AB(I) * UW(*,I)   '  X EQUATION
   10 P(*,3,I) = RKSQR(I) * P(*,3,I) - A(*) * UW(*,I)    '  Z EQUATION
     *CALL SHIFT2(P, ZSHIFT(*,RKSTEP+1))                 '  SHIFT Z BACK
     *DO 9 I = 1, 4                              '  RESTRUCTURE DATA FOR PASS 1
    9*CALL WSWAP(P(*,I,1))
     *RETURN
     *END
```

```
C*****************************************************************************
C*     TRANSFORM AND DIFFERENTIATE STRESSES IN X-Y PLANE AND FORM INERTIAL    *
C*     PART OF TIME DERIVATIVES.                                              *
C*****************************************************************************
      *SUBROUTINE PHASE3(P, SCALES, FILTER)
      *DIMENSION P(*,4,64)
      *COMMON /SCALES/ A(*), B(*), AB(*)
      *COMMON /FILTER/ EAX(*), EBY(*), EABZ(*)
      *EXTERNAL FFT, DIFF, SHIFT2
      *CU INTEGER I, Z, LSKIP, RKSTEP
      *CU REAL BY, ABZ, DELTAT
      *EQUIVALENCE (1,I), (2,Z), (24,LSKIP), (3,DELTAT), (48,BY),
      *           (47,ABZ), (4,RKSTEP)
      *COMMON /SCRTCH/ FPART(*), FACTOR(*)
      *COMMON /ALIAS/ RNXYZ(*,3), XSHIFT(*,5), YSHIFT(*,5), ZSHIFT(*,5)
       LSKIP = 4
      *DO 2 I = 1, 4
     2*CALL FFT(P(*,I,1))                    '  Y TRANSFORM
      *CALL SHIFT2(P, YSHIFT(*,RKSTEP))      '  SHIFT Y-COORDINATES BACK
      *CALL DIFF(P(*,2,1))                   '  D(UV)/DXDY
      *CALL DIFF(P(*,4,1))                   '  D(VW)/DZDY
       ABZ = AB(Z)
       FPART(*) = DELTAT * EAX(*) * EABZ(Z)
      *DO 1 I = 1, 64
       BY = B(I)
       FACTOR(*) = FPART(*) * EBY(I)     '  H * INTEGRATING FACTOR (VISCOUS)
       P(*,1,I) = FACTOR(*) * (P(*,1,I) - BY * P(*,2,I))          ' H * DU/DXDT
       P(*,2,I) = -FACTOR(*) * (A(*) * P(*,2,I) + ABZ * P(*,4,I)) ' H * DV/DYDT
     1 P(*,3,I) = FACTOR(*) * (P(*,3,I) - BY * P(*,4,I))          ' H * DW/DZDT
      *RETURN
      *END
```

```
C*****************************************************************************
C*    ADD VELOCITY INCREMENT DUE TO PRESSURE GRADIENT TO SATISFY CONTINUITY.  *
C*    INVERSE STRAINS IN CONTINUITY CONDITION ARE IN ARGUMENT SCALES          *
C*****************************************************************************
      *SUBROUTINE PRESSR(P, SCALES)
      *DIMENSION P(*,4,64)
      *COMMON /SCALES/ A(*), B(*), AB(*)
      *CU INTEGER I, Z
      *CU REAL BY, ABZ, RKYS, RKZS
      *COMMON /SCRTCH/ CON(*), PRESS(*)
      *COMMON /KSPACE/ RK(*), RKSQR(*)
      *COMMON /KSPLIT/ RKZ(*), RKZSQR(*)
      *EQUIVALENCE (2,Z), (48,I), (46,BY), (45,ABZ), (44,RKYS), (43,RKZS)
       ABZ = AB(Z)
       RKZS = RKZSQR(Z)
       CON(*) = A(*) * RKSQR(*) + ABZ * RKZS
      *DO 1 I = 1, 64
       BY = B(I)
       RKYS = RKSQR(I)
       P(*,4,I) = CON(*) + BY * RKYS                 '  WAVE # SQUARED
       PRESS(*) = (A(*) * P(*,1,I) + BY * P(*,2,I) +   -'  CORECTION POTENTIAL
      *            ABZ * P(*,3,I)) / P(*,4,I)
       P(*,1,I) = P(*,1,I) - RKSQR(*) * PRESS(*)      '  DU/DX + DP/DXDX
       P(*,2,I) = P(*,2,I) - RKYS * PRESS(*)          '  DV/DY + DP/DYDY
     1 P(*,3,I) = P(*,3,I) - RKZS * PRESS(*)          '  DW/DZ + DP/DZDZ
      *RETURN
      *END
```

REFERENCES

1. Orszag, S. A.: Numerical Methods for the Simulation of Turbulence. Physics of Fluids Supplement II, 1969, p. 250.

2. Batchelor, G. K.: The Theory of Homogeneous Turbulence. Cambridge University Press, 1953.

3. Ferziger, J. H.: Large Eddy Numerical Simulations of Turbulent Flows. AIAA J., vol. 15, no. 9, Sept. 1977, pp. 1261-1267.

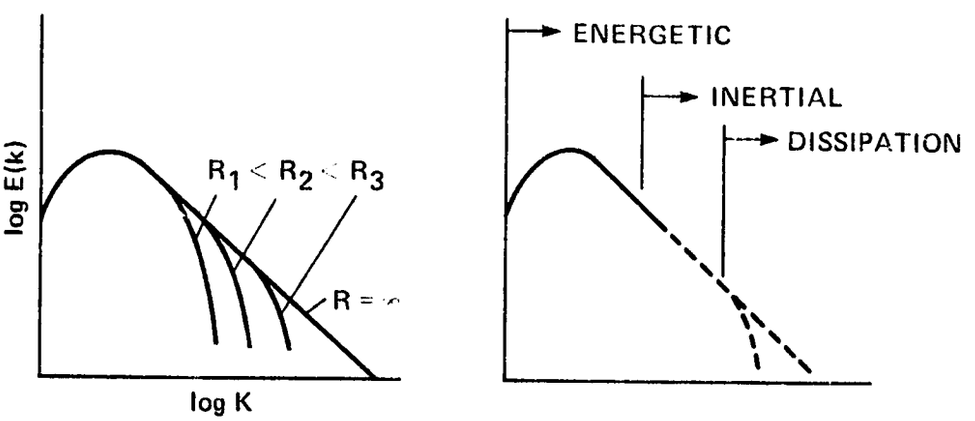


Figure 1.- Scales of motion.

NUMBER OF MESH CELLS	262144	$(=64^3)$
DEPENDENT VARIABLES	786432	$(=3 \cdot 64^3)$
DATA BASE	2.62×10^6	$(=10 \cdot 64^3)$
FFT'S PER STEP	376832	$(=4 \cdot 23 \cdot 64^2)$
COMPUTER TIME PER STEP	20 sec	(REAL TIME)
COMPUTER TIME PER RUN	10 TO 30 min	(REAL TIME)

ALGORITHM

SPATIAL RESOLUTION	SPECTRAL (ALIAS-DAMPED)
TEMPORAL RESOLUTION	RUNGE-KUTTA (FOURTH-ORDER)

Figure 2.- Simulation program.

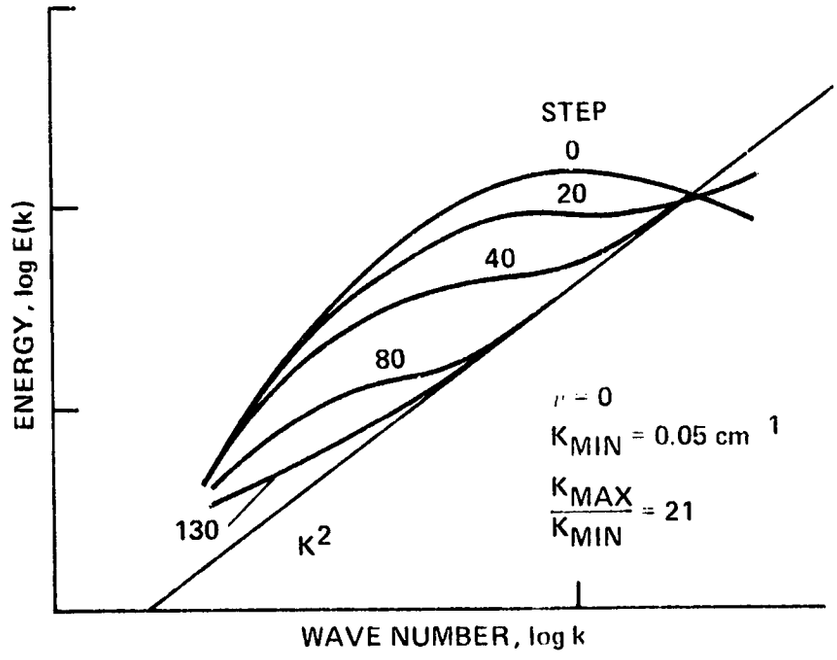


Figure 3.- Evolution toward equipartitioned energy in inviscid energy-conservative simulation.

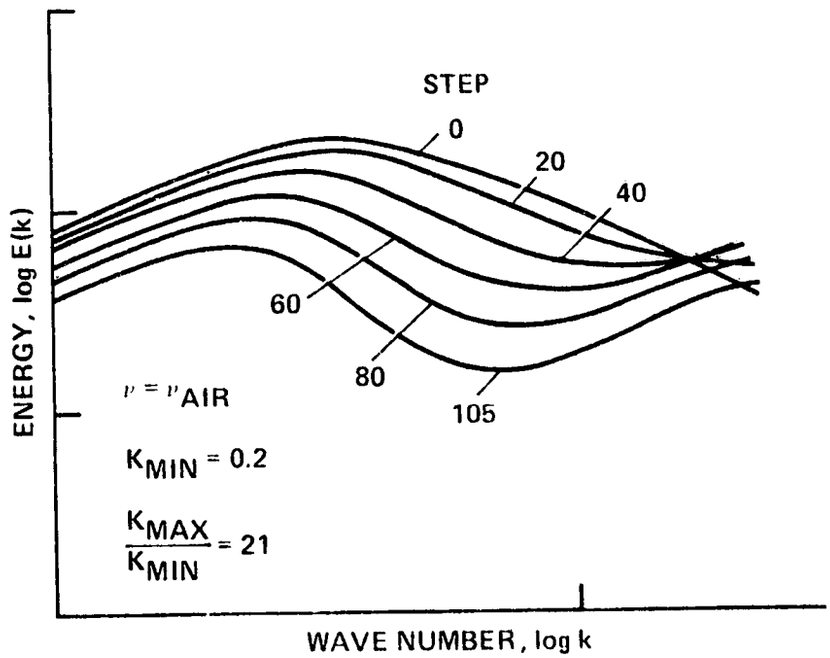


Figure 4.- Effect of viscous dissipation on computed energy spectrum.

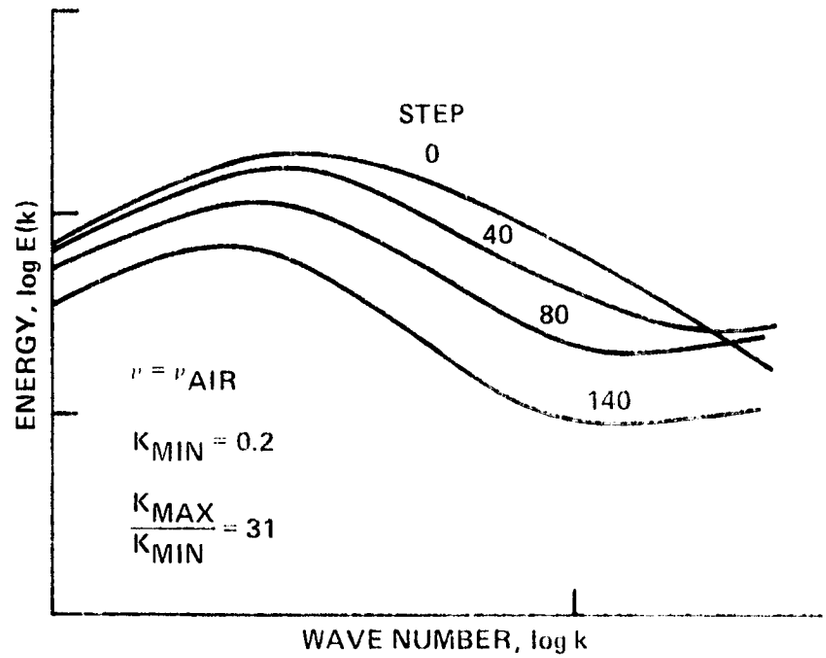


Figure 5.- Effect of higher spatial resolution on computed energy spectrum.

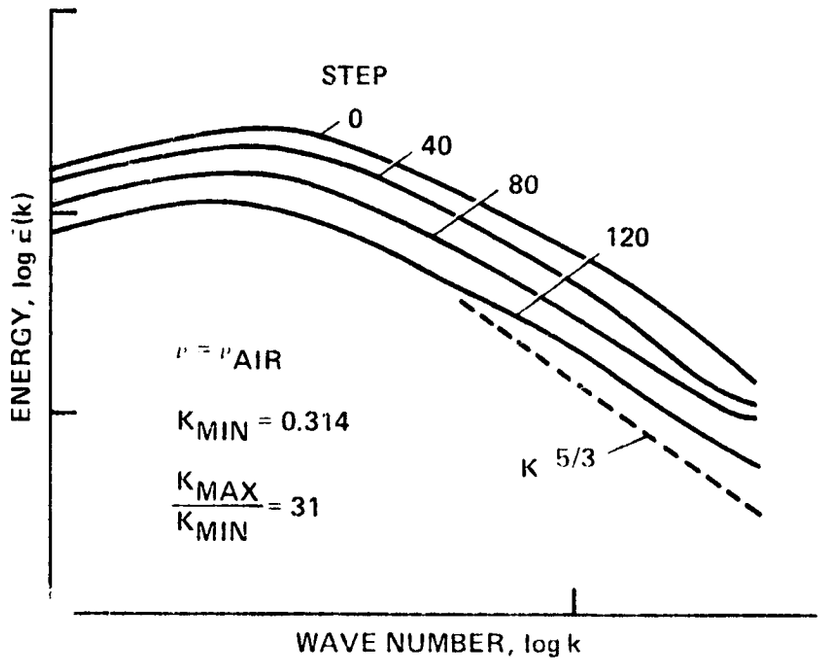


Figure 6.- Effect of scale change on computed energy spectrum.

1. Report No. NASA TM-73,203	2. Government Accession No	3. Recipient's Catalog No
4. Title and Subtitle AN ILLIAC PROGRAM FOR THE NUMERICAL SIMULATION OF HOMOGENEOUS INCOMPRESSIBLE TURBULENCE		5. Report Date 6. Performing Organization Code
7. Author(s) Robert S. Rogallo		8. Performing Organization Report No A-6899
9. Performing Organization Name and Address NASA Ames Research Center Moffett Field, Calif. 94035		10. Work Unit No. 505-06-12 11. Contract or Grant No.
12. Sponsoring Agency Name and Address National Aeronautics and Space Administration Washington, D.C. 20546		13. Type of Report and Period Covered Technical Memorandum 14. Sponsoring Agency Code
15. Supplementary Notes		

16. Abstract

An algorithm and ILLIAC computer program, developed for the simulation of homogeneous incompressible turbulence in the presence of an applied mean strain, are described. The turbulence field is represented spatially by a truncated triple Fourier series (spectral method) and followed in time using a fourth-order Runge-Kutta algorithm. Several transformations are applied to the numerical problem to enhance the basic algorithm. These include

1. Transformation of variables suggested by Taylor's sudden-distortion theory
2. Implicit viscous diffusion by use of an integrating factor
3. Implicit pressure calculation suggested by Taylor's sudden-distortion theory
4. Inexpensive control of aliasing by random and phased coordinate shifts

17. Key Words (Suggested by Author(s))	18. Distribution Statement		
Turbulence Numerical Spectral	Unlimited STAR Category - 34		
19. Security Classif. (of this report) Unclassified	20. Security Classif. (of this page) Unclassified	21. No of Pages 34	22. Price* $4.00

*For sale by the National Technical Information Service, Springfield, Virginia 22161